EXECUTIVE SUMMARY

APPARATUS AND METHOD FOR SUPPLYING ELECTRONIC CONTENT TO NETWORK APPLIANCES

COPYRIGHT NOTICE

INVENTORS

Jack B. Strong, John N. Lehner, Jonathan J. Kleid, and Vivek Patel

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority under 35 USC § 119(e) from Provisional Application Serial No. 60/212,147, entitled "Apparatus and Method for Supplying Electronic Content to Network Appliances," filed on June 16, 2000, which is incorporated by reference herein.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a simplified diagrammatic representation of an embodiment of the invention.

Fig. 2 is a simplified illustration of a progressive rendering technique employed in an embodiment of the invention.

1. Executive Summary

Bluelark's Technology Transforms the Web for PDAs

Bluelark Systems has developed a "Rapid Transformation Server" and a browser that combine to give users of wireless PDAs fast, easy access to all types of web pages for the first time. The server transforms web content into a streamlined format and optimizes it for display on a small screen, then delivers it to the PDA where it is rendered by Bluelark's "Blazer" browser. This allows users to view content created in any major format (including HTML, WML, Web Clipping and packaged "Channels") through a single browser, instead of having to use separate applications for each one as at present. Web pages load *much* faster on Blazer than on any other PDA browser because Bluelark's proprietary content streaming technology eliminates the bottlenecks found in other systems.

We Will Freely Distribute the Browser to End-Users, ...

We will aggressively promote free downloads of Blazer to stimulate usage. We expect to gain rapid adoption and "mind share" because there is a widespread need for mobile web access on PDAs, and existing products have serious deficiencies.

... Sell the Technology to ISPs and Portals, ...

We will sell the server-based transformation technology to Wireless Internet Service Providers (WISPs) and wireless portals, along with the right to distribute OEM versions of Blazer that are configured to use the customers' own proxy server and start page. Reactions by these companies to pre-release versions of Bluelark's technology have been unanimously positive. The transformation process also works well with Microsoft's Pocket IE, and we will extend it to other PDA browsers as necessary.

... And License the Platform for Custom Applications

Bluelark's technology is the ideal platform for thousands of custom wireless PDA applications, ranging from Sales Force Automation to Electronic Banking. We will license both the server and the browser components to developers in each vertical market.

No Competitor has Comparable Technology

Bluelark has a unique combination of technologies, including:

- On-the-fly HTML to WML transcoding
- On-the-fly content transformation for optimal display on a PDA screen
- Proprietary "Rapid Streaming" technology, which delivers content in a fraction of the time taken by other PDA browsing systems
- A Palm OS browser that renders a superset of WML, with Bluelark-specific extensions to improve the browsing experience

Various other companies have fragments of this technology chain (e.g. AvantGo and PumaTech both have HTML browsers for the Palm OS), but Bluelark has a 6-12 month lead in delivering a complete, viable WML/WML browsing solution for PDAs.

We Require Funding to Grow our Team and our Infrastructure

Development to date has been funded by the founders. Bluelark needs additional capital to hire sales, engineering and customer support personnel, to expand our server-side infrastructure and to purchase encryption technology for secure commercial transactions.

Bluelark has Strong Technical and Management Experience in this Field

Bluelark has unrivalled experience in delivering information services to mobile computers. Bluelark's core team collaborated – among other projects – on the design and development of the qRx software for ePocrates Inc., which is by far the dominant healthcare application for PDAs with 60,000 physician users as of May 2000.

Bluelark has 7 employees as of 6/5/00: 5 software engineers, 1 business development manager and 1 UI/graphic design specialist. We are actively recruiting to grow our engineering and business development teams.

SERVER SIDE

Project Overview

The goal of this project is to produce an HTTP proxy server with support for modules that can perform arbitrary transformations on requested files before forwarding them to the user agent. The base server will include modules for converting among several popular document markup languages and reducing the file size of common web image formats. In this document, we will examine the market opportunity and assess the key issues that will affect the outcome of the project.

Opportunity Statement

Until recently, the vast majority of internet users accessed the World Wide Web via personal computers and workstations. Web designers could safely assume that the devices accessing their sites would have similar display capabilities, processor speed, and connection bandwidth. With careful design, it was possible to produce a single version of a web site that displayed satisfactorily in nearly all browsers. Unfortunately, this is no longer possible. The number and variety of so-called "network appliances" has exploded. Millions of internet users now log on via set top boxes, personal digital assistants, and cellular telephones. Analysts predict that by the year 2002, more people will access the web via network appliances than via personal computers. Now, web designers must take into account that browsers accessing their sites may display just a few words of text at a time, or they may display hundreds. Other browsers have no display at all and instead read the content to the user through a telephone. Processor speeds and bandwidth may vary by an order of magnitude. To make matters worse, different devices support different sets of file formats, often with no intersection at all.

Web content providers that need to support users on all these devices have no choice but to provide multiple versions of their sites. Amazon.com currently has three versions of its site: the conventional HTML version, an HDML version for display on first generation cell phone browsers, and a Web Clipping version for the PalmVII. A fourth version is under development for WML capable cell phones. Only content providers with the deepest pockets can afford to author so many different versions of their sites. Even for giants like Amazon.com, this approach is clearly not scalable as the variety of network appliances continues to increase.

As with many problems in computer science, this dilemma can be at least partially solved by adding a level of indirection. In internet terms, the level of indirection between a browser and a web server is called a proxy server. The model is simple. When a browser needs to retrieve a document from the web, it sends a request to the proxy server using the browser's native protocol. The proxy then retrieves the document from the server. If the document happens to be in a format that the browser is capable of displaying, the proxy will simply forward it along. This is where the process ends for most proxy servers today.

Room for Improvement

We propose to add a critical new feature. If the document is not in a format that the browser is capable of displaying, the server will attempt to transform it into an acceptable format. For example, if the server sends a document marked up in HTML, and the browser only supports WML, the proxy can run the document through an HTML to WML translation module. This model can be extended beyond simple file format

translations to encompass display optimizations as well. For example, if the server sends a 256 color GIF image, and the device supports only 4 shades of gray, then the server can reduce the bit depth of the image, thereby saving the device the processing overhead of doing the conversion itself and reducing the file size to optimize transmission speed over a low bandwidth connection. The entire conversion process is transparent to both the web content developer and the browser user.

State of the Art

AOL developed the first non-academic transforming proxy server to use with the service's proprietary web browser in 1995. The primary purpose of the server was to save precious network bandwidth by stripping comments and unsupported markup from HTML documents and converting GIF's and JPEG's to a highly compressed format before forwarding them to the client. Other proprietary browser manufacturers followed suit, developing proxy servers to compress, and in some cases reformat, web content. WebTV developed such a server for its set top box. AvantGo and ProxiNet developed similar servers for their PDA based browsers. The most flexible proxy server of this genre is Spyglass's Prism, which supports their embedded browser on a variety of platforms.

Other vendors have taken the opposite approach, developing web server plug-ins which allow content providers to create multiple versions of their web site more easily. IBM's WebSphere Transcoding Publisher is a prime example. OnlineAnywhere offered a similar product until they were acquired by Yahoo in February.

To our knowledge, there are no commercial grade generic HTTP proxy servers which support content transformation.

Competitive Advantage

Our proxy frees end users (both content producers and content consumers) from concerning themselves with the proxy. The products mentioned above all require either the content provider or browser developer to maintain the proxy. This includes not only maintaining a server farm, but also keeping the transformation software up to date on all the latest file formats and device capabilities. By dissociating the transformation engine from both the server and client, our project will allow a third-party application service provider to take over the task of maintaining the proxy. The software can be upgraded to support new devices and formats transparently to both the content provider and the end user. We believe that this degree of transparency is crucial to widespread adoption of proxy-based transformation technology.

Additionally, our team has considerable experience developing scalable network servers and in transforming content for devices with limited display capabilities. We recently applied this experience to develop a drug database for the Palm which automatically updates itself via the internet. The application is currently in active use by over 30,000 physicians.

Other features

Dynamic construction and configuration of document transformation streams based on user-agent profiles stored locally on the proxy server, provided by the user-agent, or

retrieved from another server based on information provided by the user-agent. The profiles specify supported content-types, display characteristics of the device, and transformation preferences, and allow the server to optimize display for user-agents whose capabilities are not known at the time the server is deployed.

Conclusion

Though there is significant risk in this project, there are also significant potential rewards. The web content transformation market is far from mature, and there is still no clear market leader. A superior product could significantly grow the market as well as capture existing market share. Even if it is not possible to generically transform content targeted for PC browsers to a form suitable for network appliances, a product which does a good job of converting among the various formats intended for devices with similar capabilities would be still be useful. For example, there are currently five formats for wireless devices: HDML, WML, PQA (also know as the Web Clipping format), XHTML Basic, and AvantGo's non-standard HTML subset. A proxy server which supports converting to and from these five formats would certainly find customers.

CLIENT SIDE

Architecture

The application will be structured as an object-oriented Palm program. One set of classes will be primarily focused on aspects of the Palm environment, such as responding to launch codes and user events. A second set of classes will be responsible for browser specific tasks, such as creating a parse tree from a WML document and displaying the data. A third set will be responsible for interfacing between OS independent browsing code and the PalmOS. And finally a fourth will be comprised of generic, reusable components such as vectors and iterators. Sub-goals of this project are to create a reusable framework for Palm application development, as well as a platform independent browser kernel. We will not sacrifice the project to achieve these goals, but we will keep them in mind during design and implementation, because these sub-goals naturally lead to clean abstractions.

Palm Specific Components

Application

- Top-most level of Palm application
- Determines whether the current operating environment is suitable for this application (i.e. make sure the OS version number is not too old)
- Handles Palm launch codes (e.g. normal launch, soft reset, beam received).
- Receives all user input; must dispatch commands to the appropriate handlers.
- An application is defined by a series of forms, so this class must manage a set of forms that are each active at different times.

Form

- Represents abstract form (the Palm equivalent to a window). Meant to be subclassed by forms with specific functions.
- Supplies UI functions common to all (or at least most) forms.
- Contains zero or more views, arranged in some form specific way.

View

- Represents abstract region for displaying some sort of content or UI.
- Not nestable (i.e. a view cannot contain another view)
- Gives forms more flexibility since views are location independent (i.e. all drawing within a view is relative to the view's coordinates)

Prefs

- Manages the storage of persistent preferences used by the application.

EventHandler

- Abstract interface for any component that can handle user events (i.e. Form, View)

Browser Specific Components

Expat

- Collection of open source C files which parse XML files.
- We will use expat as a shared library to perform parsing, and will subsequently build a parse tree based on its output.

XMLParser

- Wrapper around expat and the parse tree (expat could be replaced by a different parser, and only this class would be affected)

- Manages the construction of the tree, using expat to build a tree composed of XMLElements and XMLAttributes.

XMLElement

- Single XML tag, with attributes and data

XMLAttribute

- Represents a single XML attribute

WMLGlyph

- Graphical representation of XMLElement.
- Based on recursive composition technique in *Design Patterns*, Chapter 1 (by the Gang of Four).
- A glyph can be a container for other glyphs, or can be a leaf.
- Glyphs draw themselves, and handle UI tasks within its boundaries.
- Examples of glyphs are images, blocks of text, and lines of text.

HTTPProtocol

- Implements HTTP Protocol for any underlying session mechanism (i.e. TCP)
- Possibly composed of public domain source code.

PalmOS/WML Browser Glue Components

WMLForm

- Subclass of Form, defines browser specific form features, such as back and forward arrows.
- Contains a WMLView

WMLView

- Displays WML content. Can be embedded in any application that implements the Form and View classes described above.
- Gives WMLGlyph a display area for layout
- Responsible for scrolling WML content

WMLSession

- Responsible for retrieving data, and supplying pages on demand to the WML browser.
- The underlying mechanisms will be pluggable, meaning regardless of whether a modem is used, or a Bluetooth module communicating to a cell phone, the rest of the browser will work in the same way.

TCPSession

- Subclass of WMLSession which uses Palm's built-in TCP stack
- Provides socket level instructions to enable HTTP

Generic Classes

Vector

- Easy to use data structure for managing groups of data

Iterator

- Iterator for vector, or any other class which represents a collection of objects

Types

- Defines commonly used types such as Point and Rectangle

Component Interface Design

Application

- *main*: entry point for application. Calls appropriate function depending on the launch code passed in. Performs suitability checks on environment.
- *eventLoop*: dispatches commands to appropriate eventHandlers, which have registered themselves. Keeps track of current handler.
- *registerHandler*: called by forms that can handle events, to tell the application that they should be used with forms of certain ID's. This is necessary because in the PalmOS, forms are labeled with ID numbers. By registering themselves, form objects become associated with the Palm's notion of a form.
- *launchNormal*: virtual function called by main when program is launched normally. Can be subclassed by applications to customize behavior.

Form

- *formLoad, formOpen, formClose:* Called by main event loop when this object is the current handler. formLoad should load resources needed to display the form, without writing anything to the screen. formOpen should draw the form itself. formClose should erase the form and release any resources.
- *draw*: Called by eventLoop (and possibly other functions) when the form's contents should be drawn/redrawn.
- *addView*: Adds new view to the form.
- *handleEvent*: Processes form specific events

View

- *handleEvent*: Processes view specific events
- *getBounds* Returns the area used by the view

Prefs

- *read*: reads prefs from storage into temporary data structure
- *write*: writes prefs from temporary data structure into storage
- Subclasses should provide assessors/modifiers for specific preferences.

EventHandler

- *doesHandleType*: Returns true if this object handles the specified event type (i.e. menu action)
- *doesHandleObject*: Returns true if this object handles the specified object (i.e. form with ID #4300)
- *handleEvent*: Actually handle the event. (Must be overridden by subclass)

Expat

- *XML_SetElementHandler*: Used to set the callback for new elements. Callback is called for each XML tag seen.
- *XML_SetCharacterDataHandler*: Used to set callback for data handler.

XMLParser

- *parse*: parses specified document, returns root element of document (of type XMLElement)

XMLElement

- *addAttribute*: adds attribute to element
- *getAttribute:* returns vector of attributes
- *getTag*: returns tag name

XMLAttribute

- assessor/modifier for name, value pair

WMLGlyph

- *intersects*: returns true if this glyph intersects specified point

- *insert*: inserts a sub-glyph into this glyph
- *activate*: called when the Palm UI element containing the glyph is activated
- *deactivate:* called when the Palm UI element containing the glyph is deactivated
- *draw:* recursively draws glyph, and all its sub-glyphs
- *getExtent:* returns bounds of glyph
- *parent:* returns parent glyph
- *children:* returns vector of children glyphs, if any

HTTPProtocol

- *sendRequest*: sends request to server, returns requested data. Uses underlying session protocol as provided on device.

WMLForm

- overrides functions of Form, adding appropriate behavior

WMLView

- overrides functions of View, adding appropriate behavior

WMLSession

- *setView*: way for view to register itself with a session.
- *goTo:* called by view to get data of passed in URL

TCPSession

- *goTo*: overridden from WMLSession
- provides underlying layer for HTTPProtocol to interact with, such as open socket, and read/write data.

Vector

- *add*: adds element to end of vector
- *insert*: inserts element into some position in vector
- *remove*: removes specified element

- *removeAll:* Empties vector
- *getElementAt:* returns specified element
- *getNumElements:* returns count of all elements.

Iterator

- *hasNext*: returns true if there are more elements
- *getNext*: returns next element

Data Structure Design

The most important data structures are the representations of the WML document. After parsing, a simple parse tree is created. The parse tree's structure is based on the notion that a child node is nested within their parent in the corresponding XML. For example, if a run of text looks like:

```
<p> <i> text </i> </p>
```

then the element representing 'i' will be a child of the element representing 'p'. Further, the text element will be a child of i. XML attributes are represented as a vector of string pairs, where each pair corresponds to a name and value.

This parse tree is independent from the glyph tree, although the two trees are similar in structure. The glyph tree has at least one glyph per parse tree element, and possibly more. For example, a block of text that spans multiple lines is represented as a single element in the parse tree. In the glyph tree, however, it will become multiple glyphs – one for each line, and one for the text block itself. This is based on the Composite pattern, described in *Design Patterns*.

Vectors are used extensively as a convenient abstraction on top of arrays. We would like to have used the STL for data structures such as this, but unfortunately it is not available on the Palm, as far as we know. We can take advantage of the situation and optimize our own data structures for use with the specific constraints we face.

Algorithm Design

The crux of the browser is the layout algorithm. It must be able to flow a wide variety of objects, and store their position in order to respond to user input (i.e. clicking on a link). It must be flexible enough to work concurrently with data download, so it must operate incrementally. It must be efficient, since computing resources are limited on the Palm. To facilitate these needs, we use the glyph tree to represent on screen information, as described in the previous section. The glyph tree is flexible, in that a block of text can be reformatted into its constituent lines at any point and redrawn, if for example an image is newly downloaded and changes the flow of the document.

The glyph draw function will be implemented in a relatively coordinate independent way. This means that a glyph will be passed a display area in which to draw, which it can treat as its own canvas. Each glyph must also calculate the sub-areas available to its sub-glyphs. The rendering algorithm is then composed of all glyph types following this general contract. As long as glyphs draw only in the space allotted, and gives their sub-glyphs adequate screen real estate, then the contract is fulfilled. Because of the recursive nature of the algorithm, considerable information can be cached. For example, the extent of a glyph can be stored, so the getExtent function will not need to examine all sub-glyphs to recompute the extent.